

DTIC FILE COPY

4

RADC-TR-90-49  
In-House Report  
March 1990

AD-A223 030



# ATN DESIGN TOOL

Sharon M. Walter and Scott B. Gregory

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.



Rome Air Development Center  
Air Force Systems Command  
Griffiss Air Force Base, NY 13441-5700

90 06 19 054

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No 0704-0188	
1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b RESTRICTIVE MARKINGS N/A		
2a SECURITY CLASSIFICATION AUTHORITY N/A			3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b DECLASSIFICATION/DOWNGRADING SCHEDULE N/A					
4 PERFORMING ORGANIZATION REPORT NUMBER(S) RADC-TR-90-49			5 MONITORING ORGANIZATION REPORT NUMBER(S) N/A		
6a NAME OF PERFORMING ORGANIZATION Rome Air Development Center		6b OFFICE SYMBOL (If applicable) COES	7a NAME OF MONITORING ORGANIZATION Rome Air Development Center (COES)		
6c ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700			7b ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700		
8a NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center		8b OFFICE SYMBOL (If applicable) COES	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N/A		
8c ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700			10 SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO	PROJECT NO	TASK NO
			62702F	5581	27
11 TITLE (Include Security Classification) ATN DESIGN TOOL					
12 PERSONAL AUTHOR(S) Sharon M. Walter and Scott B. Gregory					
13a TYPE OF REPORT In-House		13b TIME COVERED FROM Jun 89 TO Jul 89		14 DATE OF REPORT (Year, Month, Day) March 1990	
15 PAGE COUNT 64					
16 SUPPLEMENTARY NOTATION N/A					
17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP			
12	05		Natural Language Processing ATN Augmented Transition Network Parsing Grammar Artificial Intelligence		
19 ABSTRACT (Continue on reverse if necessary and identify by block number)					
<p>An Augmented Transition Network (ATN) is one of the most widely used formalisms for parsing Natural Language sentences. This report describes the ATN Design Tool, a tool that allows a user to graphically design ATN grammars and graphically demonstrate their use for parsing sentences. Online assistance for every aspect of creating a grammar and lexicon, including provisions for saving them to files, is available within the Tool.</p>					
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS			21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a NAME OF RESPONSIBLE INDIVIDUAL SHARON M. WALTER			22b TELEPHONE (Include Area Code) (315) 330-3577		22c OFFICE SYMBOL RADC (COES)

DD Form 1473, JUN 86

Previous editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

# ATN DESIGN AND DEMONSTRATION TOOL

## TABLE OF CONTENTS

	<u>Page</u>
I. Introduction.....	1
A. What is an ATN?.....	1
B. The ATN Tool.....	1
C. Loading the ATN Design and Demonstration Tool.....	2
II. User's Manual.....	4
A. The User Interface.....	4
B. Use of the Tool to Develop an ATN.....	12
C. ATN-Specific Functions and Symbols.....	19
D. "Running" a Parse .....	20
E. Saving an ATN to a File.....	22
F. Reading an ATN From a File.....	23
III. Design and Implementation.....	24
A. Windows.....	24
B. Structures: Arc and Node Structures.....	25
C. Global Variables.....	28
D. Window Activities.....	28
E. Arc Processing.....	37
F. Extensions to the System.....	38
 <u>Appendices</u>	
A. Definitions of Terms.....	39
B. System-Defined Arc Types.....	40
C. System-Defined Actions.....	41
D. System-Defined Forms.....	42
E. A Sample ATN.....	43
F. Command Index.....	50
 Bibliography.....	 53

# ATN DESIGN AND DEMONSTRATION TOOL

## I. Introduction

### A. What is an ATN?

An Augmented Transition Network (ATN) is one of the most widely used formalisms for parsing Natural Language sentences. James Thorne, Paul Brately, and Hamish Dewar [Thorne, Brately, and Dewar; 1968] are credited with the development of the ATN mechanism. Its use was demonstrated and popularized in 1970 by William Woods' LUNAR system. LUNAR, using a large database provided by NASA, answered questions about mineral samples brought back from the moon.

An ATN consists of a set of nodes which represent states of a parse, and arcs which represent transitions between the states and connect the nodes into networks. These networks of nodes are "augmented" with 'conditions' (sometimes called 'tests') and 'actions' attached to the arcs, and 'registers' which can be used to store intermediate results of a parse. The conditions associated with each arc determine whether or not a transition may be made to the next node. Actions associated with an arc are executed when a transition is made through it. An input sequence to an ATN is deemed to be "acceptable" (for example, a grammatical sentence) if, beginning at the unique start state, an analysis of the input leads to a final state of the network. A full description of the ATN formalism can be found in [Bates; 1978], and in the preface and assemblage of papers in [Bolc; 1983].

### B. The ATN Tool

The ATN Design and Demonstration Tool was designed to assist in the development and testing of ATN grammars and dictionaries. A graphical display of ATN networks allows users to see the grammar as it is developed and to demonstrate the sentence parsing process as it unfolds. The ATN parser included in the Design Tool uses depth-first chronological backtracking.



Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	
Justification	
By Codes	
Distribution	
Statement of Work	
Contract Number	
Project Number	
A-1	

Grammars and dictionaries developed with the Design Tool can be saved to, and read from, files for future use. Other formats for writing information to files may be user-developed to enable grammar transition to other software systems.

A sample ATN and a dictionary, configurable within the Tool, and derived from [Winograd; 1983], are included as Appendix E of this manual. They will be automatically available by mouse-selecting READ FROM FILE on the toplevel menu of the Design Tool and responding to its request for a filename by entering a carriage-return.

[Bates; 1978] has provided the principal guidelines on requirements for the ATN Design and Demonstration Tool.

### **C. Loading the ATN Design and Demonstration Tool**

The ATN Design and Demonstration Tool runs on a Symbolics<sup>tm</sup> Lisp Machine in Software Release 6.1. The software has been saved as a carry-tape. To load it:

Step 1. Place the tape into the tape reader and type at the console: *(tape:carry-load)*

Step 2. Respond to the request for the name of the machine having the tape with a carriage-return if the tape reader is on the machine connected to the console. Otherwise, type in the host machine's name.

Step 3. All of the files on the tape will be listed on the console screen. Respond "Y", for yes, to the query of whether to load the files.

Step 4. The name of the first file and a proposed new name, taking the current host name into account, will be displayed with a question as to whether it should be loaded. If the proposed name is correct (for example, it has the correct host and subdirectory name), type "A". The file will be loaded as specified and all further files will be similarly loaded.

Step 5. If the proposed name is not correct, type "O" and, at the prompt, the correct filename. At the next request type "A" and all other files will be similarly loaded.

Two files must be created. The contents of "sys>site>atn-tool.translations" on the host machine should be (include quotes):

***(fs:make-logical-pathname-host "atn-tool")***

***(si:set-system-source-file "atn-tool" "atn-tool:atn-tool;atn-tool-system")***

From this point whenever you want to load the ATN Design Tool, type: ***(Make-System 'atn-tool)*** . Press the <SELECT> key and "A" when instructed by a message on the screen to do so.

## II. User's Manual

### A. The User Interface

#### 1. The Top-Level Window

The top-level window of the ATN Design and Demonstration Tool, when the Tool is initially loaded onto the Lisp Machine, covers the entire screen (see Figure 1).

The scrollable window on the left-hand side of the screen is the Network Menu. It will display the names of ATN networks defined within the system. The window is scrolled by bumping the mouse indicator against the bottom or top of the window.

The Register Menu at the right-hand side of the screen, is a scrollable window for displaying the names of global registers that have been defined. New registers have the value Nil by default.

In the Command Menu at the bottom-right of the screen, the possible choices for selection are: ADD A REGISTER, DEFINE A NETWORK, DICTIONARY, READ FROM FILE, WRITE TO FILE, HELP!, and EXIT ATN TOOL. The small menu at the bottom-left of the screen, with the command DELETE TEMPORARIES is also considered part of the Command Menu.

The Display Window appears in the center of the screen. This window will contain the network windows and help text.

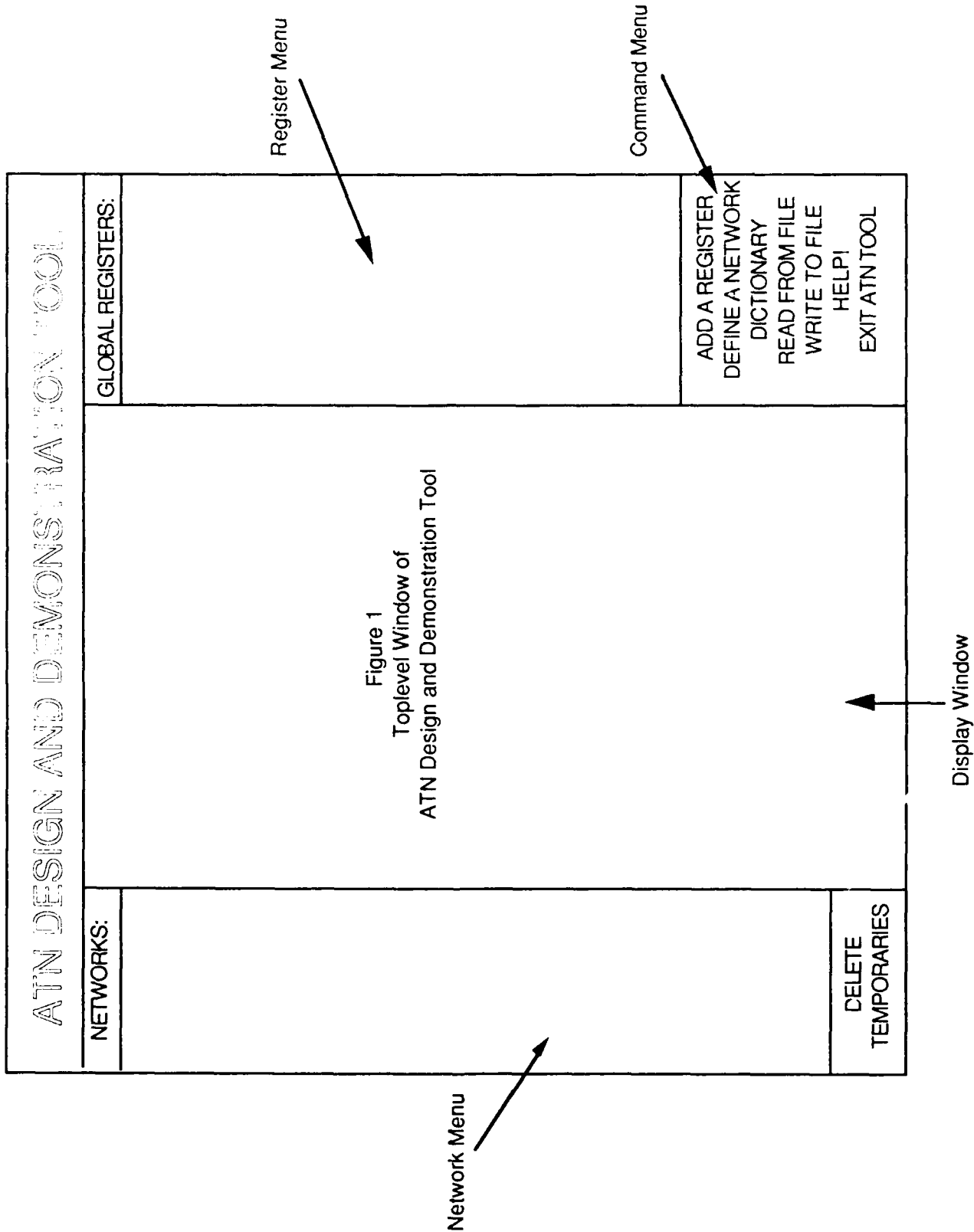


Figure 1  
ATN Design and Demonstration Tool

## 2. Development Windows

A Development Window appears on the screen when a new ATN network is being defined (user mouse-selects DEFINE A NETWORK in the top-level Command Menu), or when a user wants to look at a previously defined network (user mouse-selects an entry in the Network Menu). Development Windows have: a work area in which a network is graphically defined, a menu selection area at the bottom, and a scrollable window at the right of the screen for listing local registers.

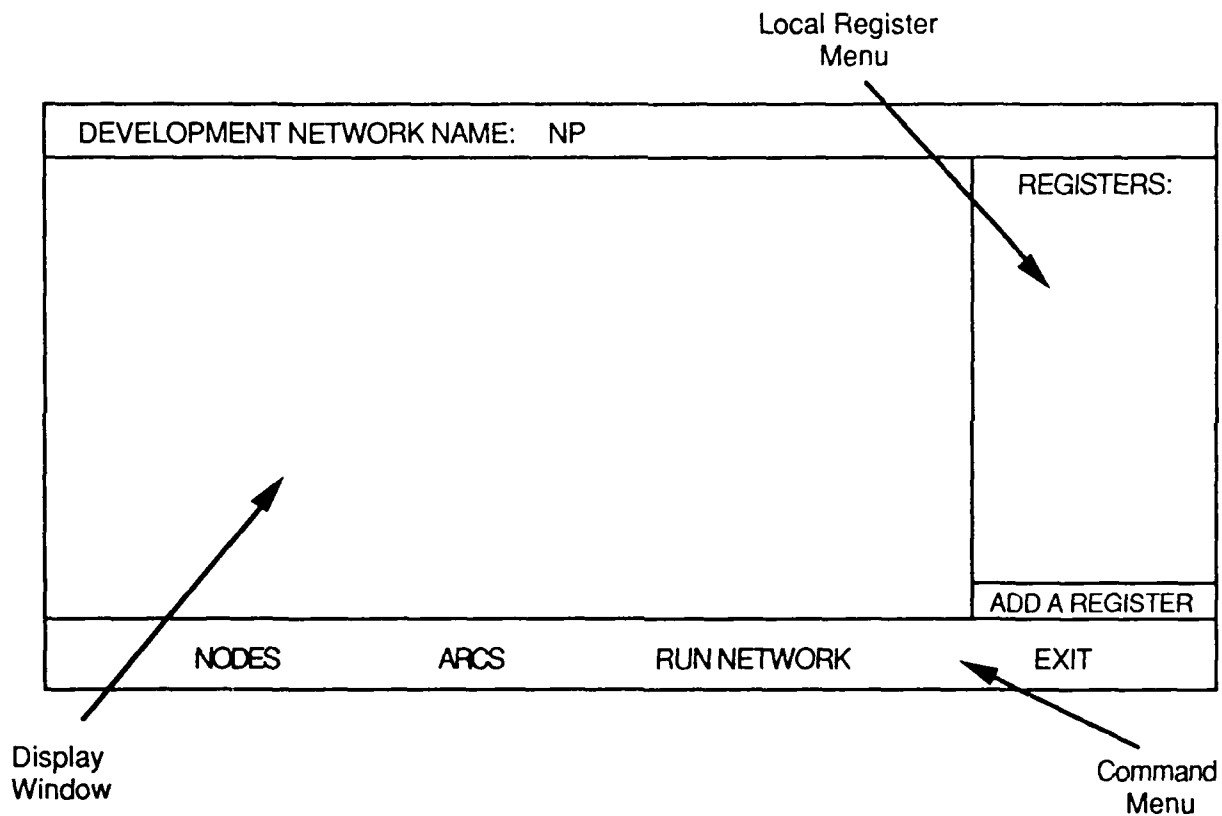


Figure 2  
Development Window

### 3. Run and Run Message Windows

After an ATN grammar and dictionary have been developed, the parsing of an input sentence can be viewed in a series of Run Windows which are created automatically during the parse. The network display, registers, and register values of a Run Window are copied from the Development Windows where the grammar was defined. The as-yet-unparsed portion of the original input is displayed at the bottom of the window. The name of each Run Window is unique, and derived from the Development Window's network name. Menu selection items in the Run Window are: MODE, STEP FORWARD, CONTINUE, and EXIT.

A trace of the parse-in-process appears in the Run Message Window above the Run Window.

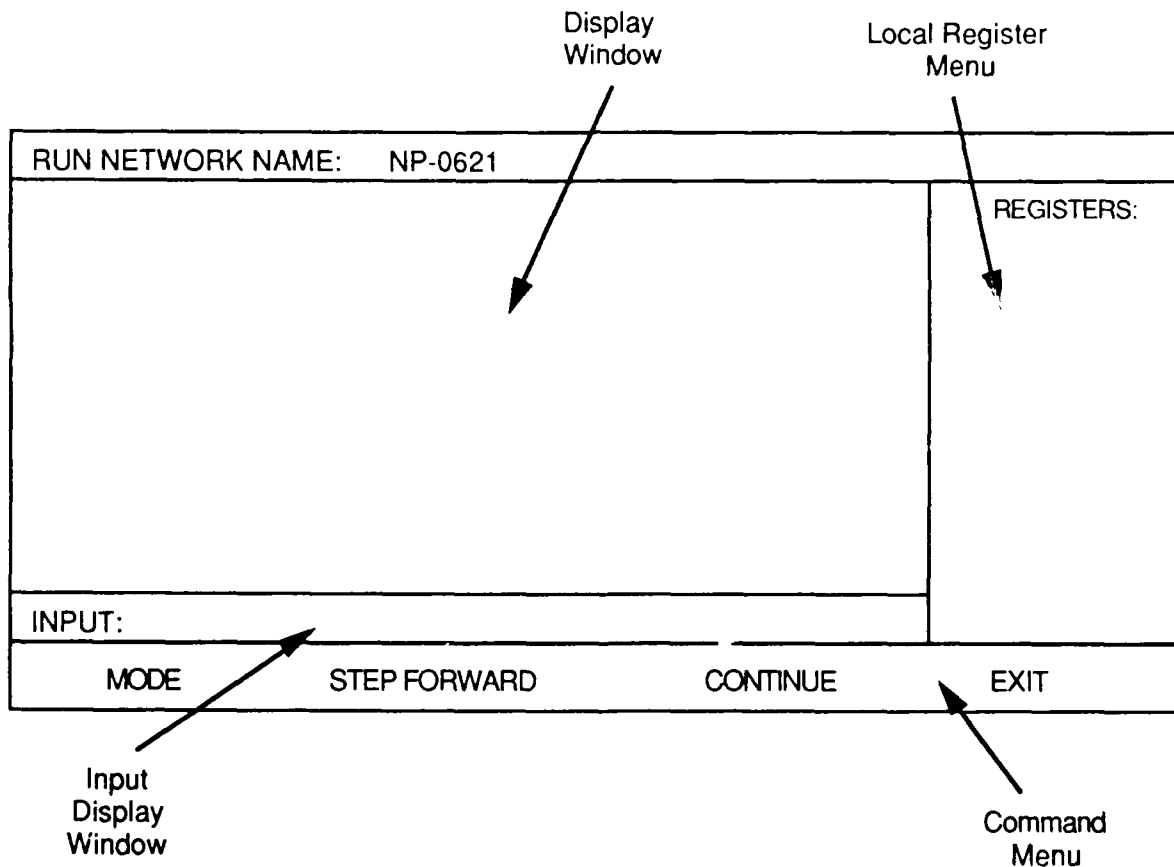


Figure 3  
Run Window

#### 4. Dictionary Window

A mouse-click on DICTIONARY in the top-level Command Menu will make the Dictionary Window appear on the screen (see Figure 4). The Dictionary Window consists of three scrollable window panes, one each for words, word senses, and features; and a Command Menu with the options CLEAR and EXIT. Dictionary words, word senses, and features can be viewed, edited, added, or deleted.

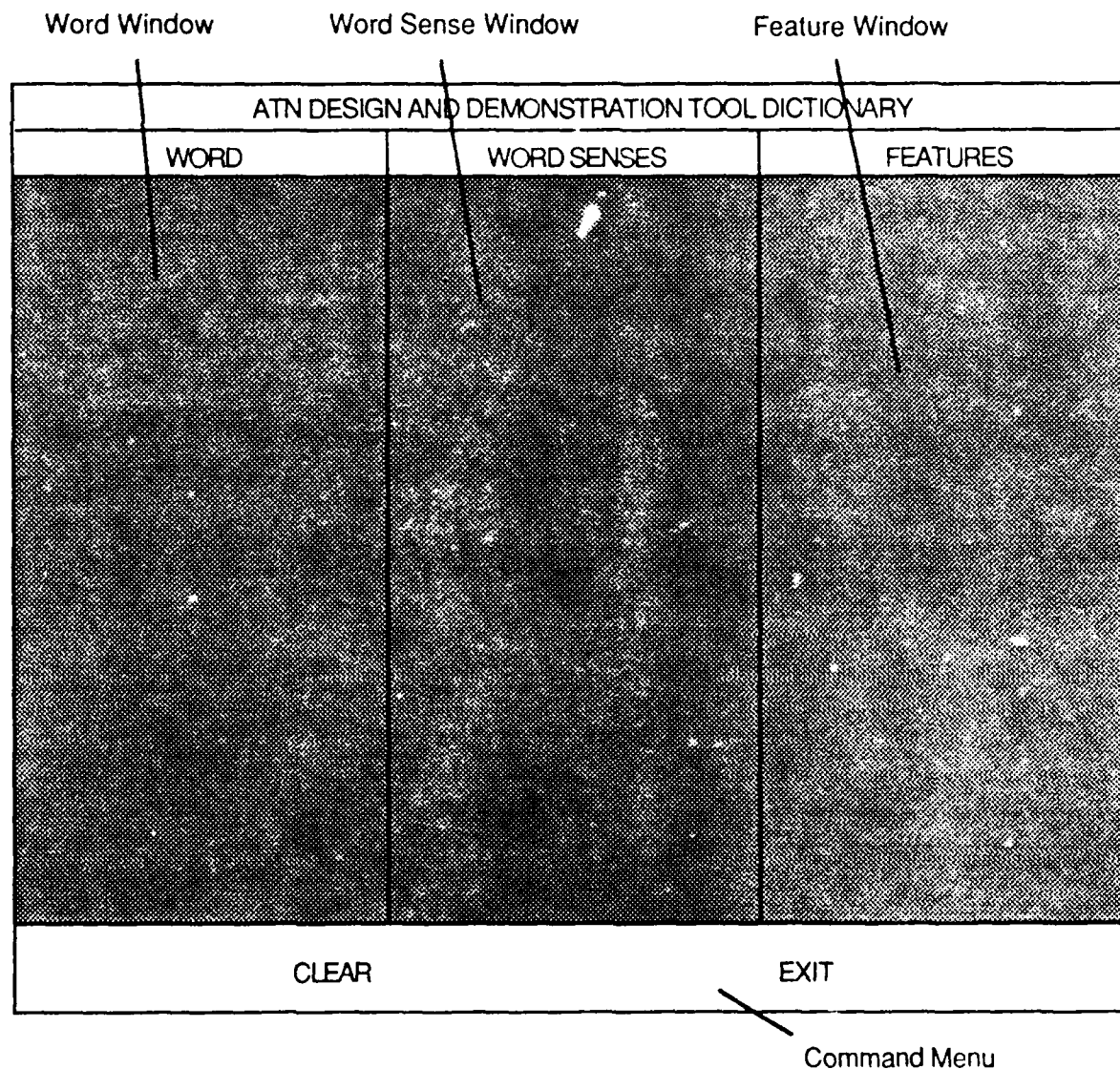


Figure 4  
Dictionary Window

## 5. Condition Window

The menu options in the Condition Window are: ADD A CONDITION and EXIT. Adding a Condition places a unique Condition identifier in the display portion of this window. When a Condition identifier is selected from the display, the user can view, edit, or delete the Condition. A value given to a Condition is a Lisp expression to be evaluated during the parsing process.

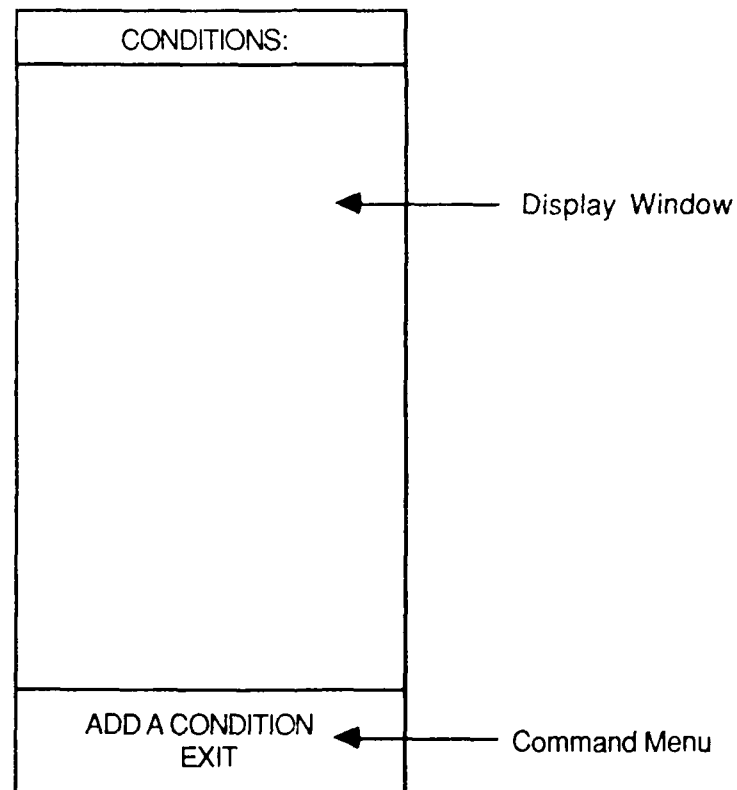


Figure 5  
Condition Window

## 6. Action and Pre-Action Windows

When Actions or Pre-Actions are to be defined for an arc, a window appears with a scrollable window and a menu of: SET A REGISTER, LEFT-ADD TO REGISTER, RIGHT-ADD TO REGISTER, LIFT REGISTER VALUE UP, ADD TO HOLD LIST, OTHER ACTION and EXIT. OTHER ACTION provides a facility for expanding system action types.<sup>1</sup> An additional menu item, SEND REGISTER VALUE DOWN, is available in the Pre-Action Window. Selecting one of the Actions from the menu places a unique identifier in the display window (ex. SETR-A048, LIFTR-A155, ADDR-A567,...). Action identifiers can be selected from the window so that the Action can be deleted or its parts specified or viewed. A fuller description of the types of Actions available within the Tool and the parts that must be specified for each type is provided in Appendix C.

Figure 6 displays an Action Window. A Pre-Action Window would have the additional SEND REGISTER VALUE DOWN Command Menu item.

---

<sup>1</sup>Assistance in defining new actions is not currently available within the Tool.

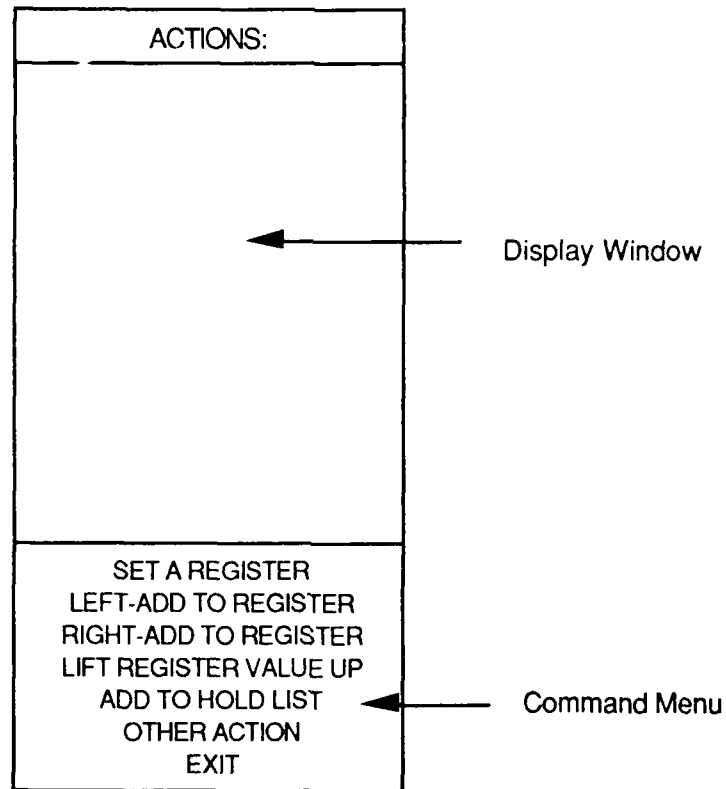


Figure 6  
Action Window

## 7. Other Windows and Menus

A number of small windows in which to type input, and menus to select from, will appear from time to time while the ATN Tool is in use. For example, an unpunctuated sequence of words is required by the system when a user selects to RUN A NETWORK in a Development Window. The sequence entered into the window becomes the input for the grammar to parse.

## **B. Use of the Tool to Develop an ATN**

Full implementation of an Augmented Transition Network requires that networks (at least one) and dictionary entries be defined. Assistance for developing both is available with the ATN Design and Demonstration Tool.

This section provides explicit, detailed instructions on the use of the ATN Design Tool. Such detail may be a hindrance for users familiar with the concept of an ATN and this manual should then be used only as a guide. The mouse-line at the bottom of the screen provides helpful guidance about the currently available options throughout the operation of the Tool.

Help files are provided for quick reference during the use of the Tool by selecting HELP! from the toplevel Command Menu.

### **1. Defining Networks**

#### **a. Create and Name a New Network**

To define a new ATN subnetwork, select DEFINE A NETWORK from the toplevel Command Menu. A small typeout window will appear with a request for a name for the network, for instance: S, NP, VP, etc. The name should be a new name, i.e. a name which has not been assigned to a network yet. After a network name has been accepted, the system will create a new Development Window and display it with the name of the network in the heading. Selecting EXIT from the Command Menu of the Development Window will remove the window from the screen. However, the name of the network will have been added to the list of defined networks displayed at toplevel in the Network Menu. The Development Window can be redisplayed by selecting its name from that menu.

#### **b. Add/Delete Nodes**

Two network nodes are already defined in the Development Window: a Start node labeled S, and a Final network node labeled F. A fully defined network will have arcs emanating from the S node, most likely to other intermediate nodes, and finally ending at the F node. All routes through a well-defined network must begin at the S node and end at the F node.

To enter the mode for adding to, and deleting nodes from, the network under development, select NODES from the Command Menu of the Development Window. To exit this mode press the middle mouse button once.

To add a node, place the mouse-indicator on the screen where the new node is to located and press the left mouse button once.

To delete an existing node, place the mouse-indicator on the node to be deleted and press the right mouse button once. The Design Tool will ask for confirmation. Press the right mouse button again to delete the node. Nodes with arcs coming from them or going to them cannot be deleted.

Other new nodes can be created, and the process intermixed with creating arcs to connect the nodes.

### c. Arc Operations

The pictoral representation of an arc is a curved line between two nodes, or a triangle when the arc emanates and terminates at the same node. The small circle that will appear somewhere on the arc is the arc's Identifier Point<sup>2</sup>.

To perform arc operations, select ARCS from the Command Menu of the Development Window. To exit from this mode, press the middle mouse button once.

Defining an arc for a network is a bit more complicated than creating a node since arcs have conditions (tests), actions, and usually other Arc type-dependent values associated with them.

### i. Add an Arc

The pictoral representation of an arc is defined by a series of mouse-clicks. Besides the start and end of an arc, which must be at existing nodes, two other arbitrary points must be identified. The first point will be the Arc Identifier Point that uniquely identifies the arc (for later editing, etc.) and another point whose sole purpose is to determine the Arc's curve.

Press the left mouse-button once to enter the mode for adding arcs.

---

<sup>2</sup>The concept of an Arc Identifier Point is unique to the ATN Design and Demonstration Tool. It is necessary for easier graphic display and handling, and has nothing to do otherwise with the concept of an Augmented Transition Network. The same is true of the arc point that is defined immediately after the Arc Identifier Point.

For each arc to be added, press the left mouse button four times. The first and last mouse clicks must be on Nodes. The second and third must not be. Each time a point is set, a '+' will be displayed. When the fourth point is set, the arc's pictorial representation will be displayed.

Pressing the right mouse-button at any time before all four points of an arc are identified will remove the parts defined to that point.

Arcs that are defined with the display's final node (labeled with an 'F') as their lastly-selected node are defined as POP arcs by default. Their Arc type cannot be changed. Other Arcs are initially defined as JUMP Arcs.

Press the middle mouse-button once to exit the mode for adding arcs.

#### *ii. Remove an Arc*

Within the Arc mode, while focusing the mouse indicator on an arc's Identifier Point, press the right mouse button. After a request for confirmation of the user's intent, the arc will be removed.

#### *iii. Defining an Arc's Features*

Within the Arc mode, while focusing the mouse indicator on an arc's Identifier Point, press the left mouse-button twice. A menu offering changes to the arc features (type, conditions, actions, and pre-actions) will appear. Most, but not necessarily all arcs require Actions and Conditions. PUSH arcs may have pre-actions specified.

(1) Changing the arc's type - If you select CHANGE ARC TYPE, a menu with all the system-defined arc types, with the exception of the current type of the arc, will be displayed. Different arc types have different information requirements in order to be fully specified. For example, in the WRD and CAT arcs a word or word category is required. The system will make requests for all of the information it needs for the new arc type. System-defined arc types and their respective requirements are listed in Appendix B.

(2) Conditions - Conditions associated with arcs are Lisp expressions that are evaluated when an arc is being considered for traversal. While running a network, if all of the arc's conditions evaluate to "non-NIL" expressions, the conditions of the arc are satisfied

and the arc is traversed. On the other hand, if at least one condition evaluates to Nil, the system will stop attempting to traverse the arc and will look for another path away from the current node.

To perform operations on conditions for an arc the ATN Tool must be in Arc mode. Then, focus the mouse indicator on the arc's Identifier Point and press the left mouse-button twice to display the arc's features menu. Select the menu item, EDIT CONDITIONS. The Condition Window that appears will allow current conditions to be viewed, edited or deleted, or new conditions to be added. Selecting EXIT from the menu at the bottom of the window will remove the Condition Window and return the system to the Arc mode.

#### -- Add a Condition

To add a Condition, select ADD A CONDITION from the menu at the bottom of the Condition Window. A unique identifier for the new Condition will appear in the window. The value of the identifier is, by default, Nil. The value of the identifier may be changed to an expression that will be re-evaluated whenever the ATN grammar is run.

#### -- View/Edit or Delete a Condition

To view, edit, or delete a condition, select its identifier from the Condition Window. A small menu will appear.

If VIEW/EDIT is selected, the Lisp expression that is the definition of the Condition is presented in a small window. Press any mouse-button, then enter a new Lisp expression to rewrite (edit) the condition. Move the mouse from the window to retain the current definition.

If DELETE is selected, the system will ask for confirmation of the user's intent and, upon receiving it, remove the Condition identifier from the window.

(3) Actions and Pre-Actions - Actions are evaluated after an arc has been crossed, before considering moves away from the next node. Pre-Actions are special actions, defined only on PUSH arcs, that are evaluated before "pushing" down to a subnetwork. As noted earlier, the set of system-defined actions and pre-actions that may be associated with an arc are listed and described in Appendix C.

To perform operations on actions or pre-actions for an arc the ATN Tool must be in Arc mode. Then, focus the mouse indicator on the arc's Identifier Point and press the left mouse-button twice to display the arc's features menu. Select the menu item, EDIT ACTIONS or EDIT PRE-ACTIONS. The Action or Pre-Action Window that appears will allow currently defined actions/pre-actions to be viewed, edited or deleted, or new actions to be added. The operations for these two windows are identical except where noted.

Selecting EXIT from the menu at the bottom of the window will remove the Action/Pre-Action Window and return the system to the Arc mode.

-- Add an Action or Pre-Action

The system-defined actions are: SET A REGISTER, LEFT-ADD TO REGISTER, RIGHT-ADD TO REGISTER, LIFT REGISTER VALUE UP and ADD TO HOLD LIST. The Pre-Action Window has the additional, SEND REGISTER VALUE DOWN. When one of these Action types is selected from the menu of the Action (Pre-Action) Window, a unique identifier, specific to that action type, is added within the window's display. That action may then be defined by editing it.

-- View/Edit or Delete an Action or Pre-Action

To view, edit, or delete an action or pre-action, select its identifier from the Action/Pre-Action Window. A small menu will appear.

If VIEW/EDIT is selected, the current values specifying the action or pre-action will be displayed one after the other. Press any mouse-button to change the current information. Move the mouse from the window to retain the current definition.

If DELETE is selected, the system will ask for confirmation of the user's intent and, upon receiving it, remove the Action or Pre-Action identifier from the window.

iv. Reordering Arcs

The order in which arcs emanating from a particular node are tried during a parse can be altered. Within the Arc mode, press the middle mouse-button twice while the mouse indicator is focused on the Identifier Point of an arc emanating from the node. All the arcs emanating from the node will be numbered in the order that they will be attempted during a parse.

Select two arcs by pressing the left mouse-button while focused on each, then press the left button again. The order of the arcs, indicated by the numbers, will be reversed.

During the arc selection process, pressing the right mouse-button once will deselect the last arc selected for reordering.

Press the middle mouse-button once to exit the Reorder-Arc mode and return to the Arc mode.

#### d. Defining Registers

Registers can be created for each ATN network and used to store intermediate results during the parsing process. Register values can only be set or accessed within the Development Window in which they are defined, with the exception of the PUSH Arc Pre-Action, SEND REGISTER VALUE DOWN, which sets a register value in the network pushed down to.

Global Registers can be defined at the toplevel of the ATN Design Tool, then accessed and set by any network during processing.

To create Global or Local Registers, select ADD A REGISTER and enter a name for the register at the system's request. The register name will appear in the scrollable pane of the Development or Toplevel Window and can be mouse-selected to VIEW/EDIT its value or to DELETE it. When the parsing process is "stepped" through (as opposed to being run continuously--see Section II. D., "Running" a Parse), intermediate values of Local and Global Registers can be checked at each pause.

## 2. Dictionary Entries

A dictionary entry is composed of a word, one or more word senses, and (for each word sense) features with feature values. For example, the word *park* may be entered with two word senses, a verb (meaning the action of *parking*) and a noun (meaning *park*, a place to take a walk). Each of the word senses will have its own features and feature values. The verb *park* could have a feature called Transitivity with the value of Transitive. The noun may have the feature Number with the value of Singular.

To perform operations on dictionary entries, select **DICTIONARY** from the toplevel Command Menu. The Dictionary Window is made up of three scrollable window panes for words, word senses, and word sense features. Operations on each of the panes are identical except where noted. When panes are shaded, the operations cannot be performed on them (for instance, a word sense for a word cannot be added until the word is identified).

Select **EXIT** from the menu to exit the Dictionary Window.

### a. Adding a Dictionary Word, Word Sense, or Word Sense Feature

To add a word, word sense, or word sense feature to the Dictionary, press the left mouse-button once on the appropriate window pane and type the item in at the system's prompting.

### b. View/Edit Values for a Dictionary Word, Word Sense, or Word Sense Feature

To view/edit the word senses of a word, the features for a particular word sense, or the value of a particular feature, focus the mouse indicator on the word, word sense, or word sense feature, respectively, and press the middle mouse-button once.

When performed on a word sense feature, this will cause a small window to appear, displaying the feature value. Press the mouse-button once to change the value. To leave the feature value unchanged, move the mouse indicator from the window.

Selecting a word or word sense in the manner described will display the word senses of the word, or the features of the word sense, in the window to the right.

GET-F takes one, two, or three (quoted) arguments. The first argument is for a feature, the second is for a particular definition (for instance, when a word has more than one definition), and the third is for a word. If only the first argument, a feature, is specified, the definition currently in focus for the current word (the value of Lex), and the current word, are assumed. The value of the feature is returned. If the first two arguments (a feature and a definition) are specified, the feature's value for the specified definition of the current word is returned. If all three arguments are specified, the value of the feature of the definition of the specified word is returned. When a given word, definition, or feature does not exist the value returned is Nil.

BUILDQ takes as arguments a special pattern, and zero or more "forms". Within the pattern the slashified at-sign (/@) can be the first member of a list in which the evaluation of each of the other members of the list results in a set of lists. The action of the slashified at-sign is to append those results together. A plus-sign (+) in the pattern will be replaced by the contents of the register that is named by the next, so-far-unused form. A pound-sign (#) causes the corresponding form to be evaluated and substitutes its value for the pound-sign. An asterisk (\*) causes the current value of \* to be substituted.

#### D. "Running" a Parse

Development Windows include a mouse-selectable option to RUN NETWORK. Input will be requested when this menu item is selected. Input should be entered as a sequence of unquoted symbols with no punctuation (ex. *the big black dog*). A parse may be started from any Development Window. For example, an NP network could be tested on *the firm green kumquats*. This facilitates the debugging of ATN subnetworks.

When RUN NETWORK is selected, a copy of the network in the Development Window is made. The copy, called the Run Network, will have a new network name: the name of the original Development Network with a distinctive suffix (ex. NP-0981). Running a copy of the network rather than the original is necessary because the same subnetwork may be called any number of times during any single parse. The menu item EXIT will remove the Run Network Window from the screen.

Each time a Run Network is made, its distinctive name is added to the Network Menu on the toplevel, main screen. The name of the new network will not be visible until the toplevel window becomes the active window. Selecting a Run Network name from the toplevel menu will cause a menu of VIEW/EDIT and DELETE to appear. Select VIEW/EDIT to bring the Run

Window back onto the display and DELETE to permanently remove the window from the system. The toplevel menu command to DELETE TEMPORARIES will eliminate the names of all the Run Networks from the toplevel Network Window, making them permanently unavailable to the user.

The course of a parse through a network is graphically displayed. When an Arc traversal is being attempted, it will blink and the arc's type will be displayed. When an Arc is successfully traversed it is darkened on the display. Backtracking may later cause the Arc to return to its normal, undarkened state.

As the parse proceeds the input can be observed as it is consumed.

### 1. Setting Modes for the Parse

Before the parsing process actually begins, the Run Window allows a user to set the Parse Mode and the Output Mode by selecting MODE from its Command Menu. The menu that appears displays the available settings for each mode. Settings are selected by mouse-selecting them. Select EXIT from the Mode Selection Menu when the modes are appropriately set.

#### a. Parse Mode

If the Parse Mode is set to SINGLE the parsing process will continue only until it finds the first successful parse of the input. Reaching the Final Node in a network indicates a successful parse so, in the SINGLE setting, the parsing process terminates the first time it reaches the Final Node. This is the default setting for the Parse Mode.

Setting the Parse Mode to MULTIPLE returns all of the successful parses. In this mode, the parser attempts to reach the Final Node as many times as it can and finally terminates at the Start Node.

#### b. Output Mode

The Output Mode determines how much textual information on the trace of the parse should be displayed in the Run Message Window during processing. Set on VERBOSE, messages are displayed about Arc traversal being attempted, successful traversals, pushes down to subnetworks, pop-ups to networks, forms returned by networks, and the network currently

being processed. The TERSE Output Mode displays only the forms returned by networks and the current network being processed.

The Run Message Window is a scroll window. The user can review the parsing of an input after the parsing process is complete.

## **2. Initiating the Parsing Process**

Once the Run Network has been created and the Parsing Modes have been set, the parsing process can proceed step-by-step, enabling the user to view the process carefully, or can be run through quickly.

Selecting STEP FORWARD from the Command Menu of the Run Window causes the ATN Tool to attempt traversal of only a single Arc. If traversal succeeds the Arc is darkened and the system is ready to proceed with the parse from the next Node in the network. At each pause between Arcs, Register settings may be inspected by selecting the register name from the Register Menu. Stepping can be repeated until the network runs to completion.

Selecting CONTINUE from the Command Menu of the Run Window cause the parse to continue through to the end. Displays of subnetworks will appear as they are pushed down to and reappear when the parse pops it up. Note that the registers cannot be examined while the parser is running in the Continuous Mode.

## **E. Saving an ATN to a File**

During pauses in the development of an ATN grammar and dictionary, or at the completion of their construction, the grammar and dictionary may be saved in a format that is readable by the ATN Design Tool or in a format of a user's design.

On the Command Menu at the toplevel of the ATN Design Tool, select WRITE TO FILE.

To save files in a format readable by the ATN Design and Demonstration Tool, enter a carriage-return to the system's request for the name of a "write" function. A name for the file to write to will be requested. If the filename given has a ".dict" suffix, only the dictionary entries will be saved. If the suffix of the filename is ".net", all of the networks and the registers will be saved. If the filename given has neither suffix, networks and dictionary entries will be saved to two separate files with appropriate suffixes.

User s familiar with the implementation of the ATN Design Tool may develop their own format for saving grammars and dictionaries to file. When a format of the user's design is to

be used, enter the name of the user defined, one-argument write function to the system's request. The function will be called with the identifier of the toplevel window as its single argument.

#### **F. Reading an ATN From a File**

To read information that had been previously defined within the ATN Design Tool back into the Tool, select READ FROM FILE from the toplevel Command Menu. A filename will be requested. The rules for the filename are the same for reading ATNs as they are for writing them: if there is a ".dict" suffix on the filename, only Dictionary items will be entered; if there is a ".net" suffix only the information defining the networks, including all the information necessary to reconstruct the network graphics, will be entered; and if neither suffix is on the filename, the system will attempt to read all of the information necessary to define an ATN grammar from two files: one with the ".dict" suffix, and one with the ".net" suffix.

### III. Design and Implementation

#### A. Windows

Users interact with the ATN Design and Demonstration Tool by moving and clicking the "mouse" and by entering information using the keyboard. The specific interpreted meaning of user actions is determined by which "window" is currently "exposed" ("unexposed" windows are either not visible, or are only partially visible and shaded).

The major windows of the Tool are the ATN Tool (the toplevel, full-screen window that appears when the system is started; see Figure 1), Dictionary (Figure 4), Condition (Figure 5), Action and Pre-Action (Figure 6), Development (Figure 2), and Run Windows (Figure 3). The latter two types of windows are constructed as needed during the use of the ATN Design Tool; Development Windows are user-created and Run Windows are system-initiated. One of each of the other windows is constructed when the Tool is started and made to appear and disappear as required.

Each of the major windows is defined by the smaller "panes" that make it up. Each file that contains a window definition begins with the definition of the large, toplevel window and is followed with the definition of the panes that are the parts of the window. Each of these files has a filename ending with "-window". For example, the file "run-window" contains the window definitions for Run Windows.

The activities of each window are defined by its "window process", that is, the toplevel function that is running when the window is exposed. The window process polls the window for a user input via the mouse and, based on the location of the mouse indicator on the window when the mouse button was pressed (and sometimes dependent on which button was pressed) proceeds with its prescribed activities. The processes for windows appear as the first function in files with a filename ending with "-process". For example, the file "run-process" contains the toplevel process for Run Windows. Specific window processes are described in a later section of this document.

Each type of window (each window "flavor") has instance variables. Instance variables are variables that are unique to windows of the defined flavor (or "type") and internal to instances of the window. For example, Development Windows have \*Network-Name\*, \*Toplevel-Frame\*, \*Network-Nodes\*, \*Registers\*, and \*Network-Arcs\* as instance variables. Instance variable values are set or retrieved by sending a message to the window.

For example, to set the *\*Network-Name\** for a Development Window (call the specific instance of the Development Window "dv-window") to NP, one would:

```
(send dv-window :set-*Network-Name* 'NP)
```

To retrieve the value of the instance variable in this case, one would send the message:

```
(send dv-window :*Network-Name*)
```

## B. Structures: Arc and Node Structures

Arcs and nodes are created by making an instance of a structure called, respectively, an Arc or a Node. Arc and Node structures are prescribed by "defstruct" operations in the "structures" file. The use of structures automatically provides facilities for creating Arcs and Nodes, and setting or accessing values for their parts. The printed representation, or name, for Nodes and Arcs is also specified by their defstructs. The printed representation for Node and Arc structures in the ATN Design Tool are as follows<sup>3</sup>:

```
(NODE name type emanating-Arcs* location)
```

```
(ARC name type conds acts type-specs to-Node* from-Node* id-point  
other-point)
```

### 1. Arcs

Parts of Arc structures that can be user-specified are its: *Type*, *Conditions*, *Actions*, *Type-Specifics*, *From-Node*, *To-Node*, *Identifier-Point*, and *Other-Point*. Additional Arc slots that are system-defined are: *Name*, *Definitions*, and *Tail-Arrays*.

An Arc is created by the structure operation "make-arc", providing any desired initial values for its parts. For example, to create an Arc, call it sample-arc, with Type "MEM":

---

<sup>3</sup>Parts of the printed representation marked with an "\*" contain only identifier(s) for the parts that they specify, not the actual value(s) for those parts.

**(setq sample-arc (make-arc type 'MEM))**

The values of the parts of sample-arc can be accessed by structure-defined functions: Arc-Type, Arc-Conditions, Arc-Actions, etc. (ie. use "Arc-" as a prefix for the name of the part of the Arc). Thus, the identifier of the Node from which sample-arc emanates can be accessed by:

**(arc-from-node sample-arc)**

Values of parts of sample-arc can be set using "setf" with the access functions. Thus, to set sample-arc to have Type WRD:

**(setf (arc-type sample-arc) 'WRD)**

a. User-Specified Arc Parts

The *Type* of an Arc is one of: POP, WRD, VIR, PUSH, CAT, or MEM.

*Conditions* is a list of Lisp expressions that are the conditions that must be satisfied before traversal of the Arc can be attempted.

*Actions* is a list of actions that are completed when the Arc is successfully traversed.

*Type-Specifics* is a list of lists. Each sublist has as its first item the name of a variable that is specific to the Arc's Type, for example, WRD arcs must have a word specified so it will have a variable "word". The second item of each sublist is the current value of the variable.

The *From-Node* is the identifier of the Node from which the Arc emanates.

The *To-Node* is the identifier of the Node where the Arc ends.

The *Identifier-Point* identifies the location of the small circle on the Arc's curve that is mouse-selected to view or alter non-graphic aspects (ex. Type, Conditions, Actions) of the Arc.

The *Other-Point* identifies the location of the third point that is specified when defining an arc.

#### b. System-Specified Arc Parts

The *Name* of the Arc is a unique name derived from Symbolics<sup>tm</sup> Get-Universal-Time function.

*Definitions* is a list of the word senses of LEX, the current word of input being processed, that have not been used in the parsing process yet.

*Tail-Arrays* is a list of points that determine the curve of the Arc.

## 2. Nodes

User-specifiable parts of Nodes are their *Type*, *Emanating-Arcs*, and *Location*. *Name* is a system-specified part.

Nodes are accessed, and the value of their parts set, similarly to Arcs.

#### a. User-Specified Node Parts

The *Type* of a Node is one of: Start, Final, or User.

*Emanating-Arcs* is a list of the identifiers of Arcs that start in the Node.

The *Location* is the location of the Node in the Development Window display.

#### b. System-Specified Node Part

The *Name* of a Node is, like Arcs, derived from the Get-Universal-Time function.

### C. Global Variables

There are a small number of global variables in the ATN Design Tool software that pertain to its windows. \*Left\*, \*Top\*, \*Right\*, and \*Bottom\*, for example, control the dimensions of the toplevel display window of the Tool; \*Mode-1\* and \*Mode-2\* hold information required for parsing modes in Run Windows.

There are also variables declared as globals in the "structures" file that are, globally, always set to Nil. Their real values are set using the "let-globally" form in the functions atn, eval-conditions, eval-actions, and eval-pop-form. Using let-globally defines these variables in the function and makes them appear to be global to all of the functions that are called by it, and to all of the functions that are called by those functions, etc.

Using the 'let-globally' form makes the values of these variables available for functions like GET-R and GET-F that are evaluated within Actions, Pre-Actions, Conditions, and POP Form statements, and whose arguments are specified by the Tool user who will not know how (or be able) to specify the identifier of a window to be used to access the global registers, local registers, dictionary entries, or word definitions currently in use. Lex and \* are available for use in the ATN Design Tool to include in Conditions, etc. These "global" variables are:

- \*Global-Regs\*
- \*Local-Regs\*
- \*Dictionary\*
- \*Forms\*
- \*Current-Definition\*
- LEX
- \*

### D. Window Activities

As noted in Section III.A, the range of possible activities available at any given time is determined by the window that is currently "exposed" (not shaded). The window "process" is the function runs in coordination with the display of a window.

It would be very difficult and lengthy to describe, in detail, the activities within each window process, or within each software file. So, the window processes will be very briefly

described and only their non-visible actions, such as variable settings, included. These brief descriptions and the well-documented code will make the method of implementation of the ATN Design and Demonstration Tool apparent, and facilitate any investigation into specifics of the system's operation.

The window process for each window loops infinitely, waiting for a "blip" (command) from the screen. It is the first symbol of the blip that is used in the window process to determine the appropriate functions to apply. In Symbolics Lisp, when an item is mouse-selected from a menu, the first symbol in the blip is ``:menu'`. When an item is mouse-selected from a scroll window, such as a Register Window or the Dictionary Window, the first item of the blip is determined by the `:print-item` message defined for the Window (see any window definition file). For example, the toplevel Network Window returns a blip with a first item of ``:network'`.<sup>4</sup>

Pressing a mouse-button when the mouse indicator is somewhere other than on a menu returns a blip whose first item is the symbol `:mouse-button`.<sup>5</sup>

Each window process description below gives the first item for each possible mouse blip within the window. These signal where the mouse indicator was when the mouse-button was pressed. For example, `(:menu)` indicates the mouse-button was pressed over a Command Menu, `(:register)` indicates the mouse-button was pressed over a Register Window.

The Run Window process is the most interesting of the window processes. The majority of the activities of the other windows consist of requesting, accepting, and storing short inputs. The Run Window process has the task of processing, and displaying the course of processing, the network. The descriptions of other window processes then, will focus mainly on the format of stored values when those values are stored as other than, simply, symbols. Each process description will end with information on activities that cause other windows to become exposed.

---

<sup>4</sup>The second item of the blip for either of these types of menus tells which item in the menu was selected. The third item tells which of the three mouse-buttons was pressed to make the selection, and the fourth item identifies the window that the item was selected from.

<sup>5</sup>The rest of the signal specifies, respectively, which mouse-button was pressed, which window the mouse was on at the time, and the X- and Y-coordinates of the location of the mouse indicator.

## 1. ATN Toplevel Window Process

Within the toplevel window, activities include those that can be selected from the command menu (:menu), the global register window (:global-register), and the network window (:network).

When global registers are created, deleted, or given values the *\*Registers\** instance variable (a list) of the toplevel window is manipulated. When a register is added, a list with the name (a symbol) of the register is put into *\*Registers\**. If a register is given a value, the value becomes the second member of the sublist. The format of *\*Registers\** is shown below. Any value, or values, may be Nil.

*\*Registers\**: ((*reg1 val1*) (*reg2 val2*) ... (*regn valn*))

*\*Current-Window\** contains the name of the Development or Run Window that is currently exposed.

The *\*Defined-Networks\** instance variable of the toplevel window contains the names and identifiers of all the Development Windows that are accessible. *\*Defined-Networks\** consists of sublists that are pairs of network name and network window identifier. Whenever a Development Window is defined, a list such as '(NP <NP-window-identifier>)' is added to the list. When any of the Development Network names is deleted from the Network Window, its corresponding sublist is eliminated from *\*Defined-Networks\**.

*\*Defined-Networks\**: ((*S <S window id>*) (*NP <NP window id>*))

Another instance variable, *\*Temporary-Networks\** is a similar list containing the names and window identifiers of only Run Windows (no Development Window sublists). When a Run Window is deleted its corresponding sublist is deleted from *\*Temporary-Networks\**.

A number of other windows are accessible from the toplevel window. Instance variables in the toplevel window definition store identifiers for the Action Window (*\*Actions-Window\**), the Pre-Action Window (*\*Pre-Actions-Window\**), the Condition Window (*\*Conditions-Window\**), the Dictionary Window (*\*Dictionary-Window\**), and the window from which the ATN Design Tool was created (*\*Initial-Calling-Window\**). Although the identifiers for each of these windows is available in the toplevel window, only the latter two are ever accessed by the toplevel window process. The Dictionary Window can be called into

view (at which point it is the active window and control is taken over by its window process) by selecting DICTONARY from the toplevel Command Menu, and the window that the ATN Design Tool was started from (usually a Lisp Listener) can be returned to by selecting EXIT ATN TOOL. The other windows are accessed from Development and Run Windows.

Development windows can be created, their identification sublist added to *\*Defined-Networks\** (see the information about the *\*Defined-Networks\** instance variable, above), and then exposed from the toplevel window by selecting to DEFINE A NETWORK.

## 2. Development Window Process

Within a Development Window, activities include those that can be selected from the Command Menu (:menu), the Register Window (:register), or the Display Window (:mouse-button).

When registers are added, deleted, or given values in a Development Window, the *\*Registers\** instance variable is manipulated in the same manner as the *\*Registers\** instance variable of the toplevel window:

*\*Registers\**: ((*reg1 val1*)(*reg2 val2*) ... (*regn valn*))

The *\*Network-Nodes\** instance variable contains a list of the identifiers for the Start and Final Nodes when a Development Window is initialized. When a new Node is created, its identifier is put on the list (or deleted from the list when the Node is deleted from the window). When an Arc is defined, its identifier is similarly placed in the list of *\*Network-Arcs\**.

If the menu item to RUN NETWORK is selected, a Run Window is defined, which is a copy of the current Development Window. The *\*Input\** instance variable of the Run Window is set according to the user's input, and the Run Window is exposed (passing responsibility for activity to it).

When the Development Window is EXITed, the toplevel window recovers activity responsibility.

### 3. Run Window Process

In the Run Window, menu activities (:menu) and register activities (:run-register) activities may be selected.

The list of Run Window instance variables includes all of those of a Development Window (\*Network-Nodes\*, \*Network-Arcs\*, etc.) *plus the items that will accurately define the current state of a parse at any point of the parsing process.*

\*Development-Window\* will hold the identifier of the Development Window that the Run Window is a copy of.

\*Toplevel-Frame\* is an identifier for the toplevel window, allowing the Run Window to access the Dictionary and other windows.

The value of \*Parse-Mode\* will be either SINGLE (the default) or Multiple, indicating user preference to find the first successful parse of the input or to find all possible parses.

The value of \*Output-Mode\* will be either VERBOSE (the default) or TERSE, determining how much textual information will be displayed in the Run Message Window during a parse.

\*Trace-Net\* will be Nil or hold an identifier for the Run Window that called the current Run Window.

\*Sub-Networks\* will contain a list of the networks that the network has pushed down to during its run.

\*Current-Node\* holds the identifier of the network node at which the parse is currently standing. Upon first exposing the Run Window the current node value will be the Start Node of the window.

\*Trace-Nodes\* will be a list of the Node in the current network that have been passed through, in reverse order.

\*Input\* will be a list of unpunctuated, unquoted symbols representing the currently unprocessed part of the user's input.

\*Arcs-To-Try\* will be a list of arc identifiers. The arcs in the list will be those that emanate from the current node (\*Current-Node\*) whose traversal has not yet been attempted.

\*Graphic-Trace-Arcs\* is a list of all the Arcs that have been successfully traversed up to the current state of the parsing process.

\*Successful-Subparse\* is initially Nil. It will be set to T if the pass through the network was successful. It is used as a signal for an upper-level network if there is one.

*\*Finished?\**, initially Nil, becomes T when the network either fails or succeeds.

*\*Continue?\** is initially Nil. It is set to T if a user selects CONTINUE from the Run Window Command Menu in order to have the parse go to its conclusion without pausing. After a step through a parse, if *\*Continue?\** is T, the next step is automatically taken.

*\*Reactions\** is a list of Lisp Expressions that will be evaluated in order to set the state of the parse back one step. While an arc traversal is being attempted, expressions are added to this list in case the parse later fails, causing the current arc to be backtracked over. When an arc is successfully crossed the contents of *\*Reactions\** is added to the *\*History-List\** and *\*Reactions\** is set to Nil.

*\*History-List\** contains a list of lists of *\*Reactions\** to be taken in case of backtracking. For each arc backed-up over, the contents of one sublist (the first one) is evaluated in order to reset the state of the parse.

*\*ATN-Parses\** is a list of the results from the most recent parse.

The cornerstone function of parsing activities is the "step-forward" function in the *atn* file. Each time this function is called, one step, or attempt at a step, through the parse is made.

If there are arcs emanating from the current node (*\*Current-Node\**) whose crossing has not yet been attempted (*\*Arcs-To-Try\**), an attempt will be made to process/traverse the first arc according to the requirements set up by its characteristics. For example, if the first arc of *\*Arcs-To-Try\** is a WRD arc the first word of the remaining input must match the Word stored in the Type-Specifics of the Arc, and all of the Conditions of the Arc must evaluate to something other than Nil. If those two factors allow, the Arc is successfully traversed, the graphic display indicates traversal, and the Actions are evaluated.

If all emanating arcs from the current node have been tried and the current node is the Final Node of the network, then the parse through the network was successful if *\*Input\** is empty, or if there is an upper level network to POP to.

If all emanating arcs from the current node have been tried and the current node is the Start Node of the network, then the parse through the network failed, or has found all possible parses of the input. If the current network was called from an upper level network, the failure of this network is signalled to the upper level.

If all emanating arcs from the current node have been tried (and the current node is NOT the Start or Final Node of the network), an arc is backtracked through.

#### 4. The Dictionary Process

Activities that can be selected within the Dictionary include Command Menu activities (:menu), activities related to words, word senses, or word sense features selected from the three windows (:items), and general activities related to pressing mouse-buttons in open areas of the windows (:mouse-button).

Instance variables *\*Toplevel-Dictionary-Window\**, *\*Dictionary-Word-Window\**, *\*Dictionary-Definition-Window\**, and *\*Dictionary-Feature-Window\** allow the windows to send and receive information about dictionary entries to one another. The word, word sense, and word sense feature portions of the Dictionary Window each have two of these instance variables, linking the Dictionary Window to the word portion of the window, the word portion to the word sense portion of the window, and the word sense portion to the word sense feature portion of the window.

The instance variable *\*Word\** is associated with the word sense portion of the Dictionary Window and the instance variable *\*Definition\** is associated with the word sense feature portion. The value of *\*Word\** is the word from the word portion of the Dictionary that is in focus. The value of *\*Definition\** will be the particular word sense that is in focus.

Within the Dictionary Window, the value of the *\*Dictionary\** instance variable is a list of lists, each sublist representing a word and its definition. The format of *\*Dictionary\** is as shown in Figure 7.

```

*Dictionary*:      (      (word1      (word-sense1-1      ( (feature1-1-1
                      value1-1-1)

                                (feature1-1-2  value1-1-2)
                                •
                                •
                                (feature1-1-m  value1-1-m)))

(word-sense1-2 • • • )
      •
      •
      •

(word-sense1-n ( (feature1-n-1  value1-n-1)
                (feature1-n-2  value1-n-2)
                •
                •
                (feature1-n-r  value1-n-r))))

(word2      • • • • ))))
      •
      •
      •

(wordz      (word-sensez-1 ( (featurez-1-1  valuez-1-1)
                            (featurez-1-2  valuez-1-2)
                            •
                            •
                            (featurez-1-t  valuez-1-t))))

(word-sensez-2 • • • • ))
      •
      •
      •

(word-sensez-x      (      (featurez-x-1  valuez-x-1)
                          (featurez-x-2  valuez-x-2)
                          •
                          •
                          (featurez-x-v  valuez-x-v))))

```

Figure 7  
\*Dictionary\* Format

## 6. Conditions Window Process

There are menu activities (:menu) and activities relating to Condition identifiers in the display window of the Conditions Window (:condition).

Recall from Section III.B. that part of an Arc structure is called Conditions. The value of the Conditions part of an Arc is a list of lists, each of which defines a Condition. Each sublist has the unique Condition name (something like COND-873) as its first member and the user-defined Condition (a Lisp expression) as its second member. A typical sublist might be something like:

```
(COND-873 (eq (getr 'subject) *))
```

## 7. Action Window Process

There are menu activities (:menu) and activities relating to Action identifiers in the display window of the Actions Window (:action).

The activities of the Action Window are very similar to those of the Condition Window. Part of an Arc structure is called Actions. When a user choses to add an Action to an Arc, a unique Action name is defined and a form representing that Action is added to the Actions part of the Arc (see the section on Structures, Section III.B.). The form is a framework for specifying the particular kind of Action that is being added. It contains the names of required parts for that kind of Action (for instance, a SETR Action requires a Register and a Form), with their values set to be Nil. Thus, when a SETR Action is added, a unique name is created (say, SETR-056) and a form is added to the Arc Actions that looks like:

```
(SETR-056
  (Act-Type 'SETR)
  (Register)
  (Form))
```

When an identifier for an Action is selected from the Action display window, values of its parts can be set. In the case of the example above, the name of a Register and a Form will be requested.

EXITing from the Actions Window returns control to the Development Window it was called from.

## 8. Pre-Action Window Process

As with the Action window process, activities selectable within the Pre-Action Window are associated with menu items (:menu) and actions (:pre-actions). Activities within the Pre-Action Window are exactly the same as Action Window activities except for the part of the Arc structure that is manipulated. The purpose of the Arc part called 'Type-Specifics' (see the section on Structures, Section III.B.) is to specify things that are particular to the type of Arc. Thus, a CAT arc should have a value set for the Category part of its Type-Specifics and a PUSH arc could (although pre-actions are not absolutely necessary) have a value set for the Pre-Actions part of Type-Specifics. So, the part of an Arc structure that Pre-Action window process activities manipulate is the Pre-Action subpart of Type-Specifics.

### E. Arc Processing

The function *step-forward* in the "run-process" file controls stepwise movement through a network, taking into account the current state of the parse, the current input, dictionary entries, and the specifics of the network definition. The file "parser" contains the functions specific to processing Arcs.

When the parsing process arrives at a new Node, the contents of the Node's Emanating-Arcs part is copied into the network instance variable \*Arcs-To-Try\*. For Arcs in \*Arcs-To-Try\* that consume input (CAT, WRD, and MEM), the Definitions part is set to the list of word senses of the current word (LEX) so that the traversal of each of those Arcs will be attempted with each of the word senses.

The parsing process then begins. Traversal of the first Arc in \*Arcs-To-Try\* is attempted, checking the Arc's requirements and characteristics of the current word of input. For example, for a WRD Arc the current word of input must match the word stored in the Arc's Word slot of its Type-Specifics, and all of the Arc's Conditions must evaluate to non-*Nil*.

If the Arc is determined to be traversable and it is a PUSH Arc, its Pre-Actions are evaluated, processing "pushes down" to a subnetwork (creating and displaying a new Run Window), and upon returning from successful traversal through the subnetwork, the PUSH

Arc's Actions are evaluated and the function *success-settings* updates the network's instance variables and graphic display to indicate success. If traversal of the subnetwork fails, traversal of the PUSH Arc fails.

If the Arc is determined to be traversable and it is not a PUSH Arc, the Arc's Actions are evaluated and *success-settings* updates the network's instance variables and graphic display (darkening the Arc to mark it as successfully traversed).

When traversal of an Arc fails, the next attempt is made. The next attempt will be with the same Arc if it is an Arc that consumes input (CAT, WRD, and MEM) and the current word has word senses that have not been tried. If all attempts to traverse the Arc fail, it is removed from \*Arcs-To-Try\* and traversal of the next Arc is attempted.

If traversal of all of the Arcs in \*Arcs-To-Try\* fail, then backtracking occurs over the Arc that was crossed to get to the current Node, the Lisp expressions in the Run Window's \*Reactions\* instance variable are evaluated (returning the network variables to their previous state), and the graphic display is updated.

After each attempted Arc traversal, whether successful or not, the state of the Run Window display and Run Window instance variables are in a stable state and ready to try the next step.

In Single Parse mode, processing terminates successfully upon first reaching the Final Node of the toplevel network, or unsuccessfully when all paths to the Final Node have been attempted and failed. Multiple Parse mode, however, terminates processing (successfully, returning all parses, or unsuccessfully) when all paths to the Final Node have been tried.

## **F. Extensions to the System**

Currently, the ordering of Word Senses, Conditions, Actions, and Pre-Actions is determined only by the order in which they were defined or edited. Users should be provided with the ability to purposefully order these items.

No morphological analysis tools are provided directly within the ATN Design Tool, and that would seem to be appropriate as it is not within the standard ATN paradigm. Presumably, good language coverage could be achieved by either preprocessing input and having networks account for grammatical "signals" (indications, for example, of word tense), or, by loading the Dictionary with all possible word forms (which is the approach taken in the sample ATN given in Appendix E).

## APPENDIX A

### Definitions of Terms

The following terms are used in this document and in the ATN Design and Demonstration Tool. The required form and meaning of each is described. Examples are also provided.

1. <actions>                      *Ex. Setting a Register is an Action*

Members of the ATN concept-specific Action types. They are described in Appendix C.

2. <category>                      *Ex. Noun*

A symbol representing the value of the Feature 'Part-of-Speech'.

3. <constit-type>                      *Ex. NP*

A symbol representing a network.

4. <form>                      *Ex. (list (GETF 'Part-of-Speech LEX) LEX)*

A Lisp expression to be evaluated at a later time, for instance when a register is set to the value of the form.

5. <list-of-words>                      *Ex. (cat dog horse)*

A Lisp list of unquoted symbols.

6. <pre-actions>                      *Ex. Set a Register in the next lower network to a specified value.*

Members of the ATN concept-specific Action types, including the PUSH arc pre-action: SENDR. The Pre-Action Definition Window assists in defining pre-actions. See Appendix C for detailed information on Action types.

7. <reg> or <register>                      *Ex. HOLD*

A symbol representing a global or local register name.

8. <tests>                      *Ex. (eq (GETR 'reg1) \*)*

Lisp expressions to be evaluated in order to determine whether an Arc traversal should be attempted. If one or more of the test (or "conditions") evaluate to Nil, the arc attempt is halted.

9. <word>                      *Ex. dog*

A symbol representing a word in the Dictionary.

## APPENDIX B

### System-Defined Arc Types.

The following types of arcs are readily available within the ATN Demonstration Tool. Other arcs may be user-defined, but it is not recommended that one do so without a thorough understanding of the system design.

1. (CAT <category> <tests> <actions>)

A CAT arc may be traversed if the current input word is of the category specified by the second element of the arc.

2. (WRD <word> <tests> <actions>)

Specifies the exact word of input which is required.

3. (MEM <list-of-words> <tests> <actions>)

Like a WRD arc except that the input word must be on the list of words given as the second element of the arc.

4. (PUSH <constit-type> <tests> <pre-actions> <actions>)

Initiates a subcall to the network which begins in the indicated state (specified by <constit-type>) and attempts to parse the current input with that network.

5. (POP <form> <tests>)

Returns control to the PUSH arc which caused the process at the current level to be invoked. A POP arc has no destination state. It marks the state that it leaves as a terminal state for some level of the network. Its second element indicates the form which is to be returned as a result of the analysis of the portion of input parsed by the current level of the network.

6. (VIR <tests> <actions>)

Checks whether a constituent of the named type has been placed on the HOLD list by a HOLD action of some previous arc.

7. (JUMP <tests> <actions>)

Specifies the state to which the transition is to be made. Nothing is consumed from the input string.

## APPENDIX C

### System-Defined Actions

Actions in an ATN construct pieces of structure and store the structures in registers. Listed here are the actions that are immediately available in the ATN Demonstration Tool. The menu item for each Action as it is displayed in the Action Window of the ATN Design Tool is also provided.

1. (SETR <reg> <form>) "SET A REGISTER"  
Causes the indicated register to be set to the value of the form.
2. (ADDL <reg> <form>) "LEFT-ADD TO REGISTER"  
Adds the value of the form to the left end of the named register.
3. (ADDR <reg> <form>) "RIGHT-ADD TO REGISTER"  
Adds the value of the form to the right end of the named register.
4. (SENDER <reg> <form>) "SEND REGISTER VALUE DOWN"  
Causes the named register in the next lower level of recursion to be initialized to the value of the form. SENDER is only appropriate as a pre-action for PUSH arcs.
5. (LIFTR <reg> <form>) "LIFT REGISTER VALUE UP"  
The inverse of SENDER. LIFTR sets the named register, in the level just above the current level of recursion, to be set to the value of the form.
6. (HOLD <constit-type> <form>) "ADD TO HOLD LIST"  
Places the indicated form on the HOLD list as a constituent of constit-type. (The HOLD list is a global list, ie. accessible at all levels.)

## APPENDIX D

### System-Defined Forms

The following 'forms' are widely used within the ATN paradigm. They are Lisp functions used in conjunction with actions on arcs.<sup>6</sup> Arguments that appear in *italics* are optional. If they are not specified they will assume default values.

1. (GET-R <register>)

Returns the contents of the specified register (a quoted symbol).

2. (GET-F <feature> <word sense> <word>)

If the word sense of the word has the specified feature, its value is returned. If a word sense is not specified, the current word sense associated with the current Arc is assumed. If the word is not specified, the value of LEX is assumed. All given arguments must be quoted symbols.

3. (GET-D <word> <all-word-senses?>)

Called with no arguments GET-D returns the current word sense of the current word of input (LEX). Given a word as its first argument (a quoted symbol), GET-D returns the first word sense of the word. Given a second argument of T, GET-D returns all word senses of the word as a list.

4. (BUILDQ <template> <form<sub>1</sub>> <form<sub>2</sub>>...<form<sub>n</sub>>)

Allows arbitrary structures to be built. BUILDQ takes as arguments a special pattern and zero or more forms. When the slashified at-sign (/@) is the first member of a sublist in the pattern, the other members of the sublist are appended together. A plus-sign (+) in the pattern will be replaced by the contents of the register that is named by the next, so-far-unused form. A pound-sign (#) in the pattern causes the corresponding form to be evaluated and its value substituted for the pound-sign in the pattern. An asterisk (\*) causes the current value of \* to be substituted.

---

<sup>6</sup> Actually, the traditional ATN function GETF has been renamed to GET-F so as not to interfere with the built-in Lisp function GETF. The other functions have been similarly named with hyphenated names for consistency.

## APPENDIX E

### A Sample ATN

An example of an Augmented Transition Network, derived from [Winograd; 1983], is detailed here. It is available within the ATN Design and Demonstration Tool and can be loaded by selecting READ FROM FILE from the Command Menu on the toplevel window and responding to the request for a filename by entering a carriage-return.

#### a. The Dictionary

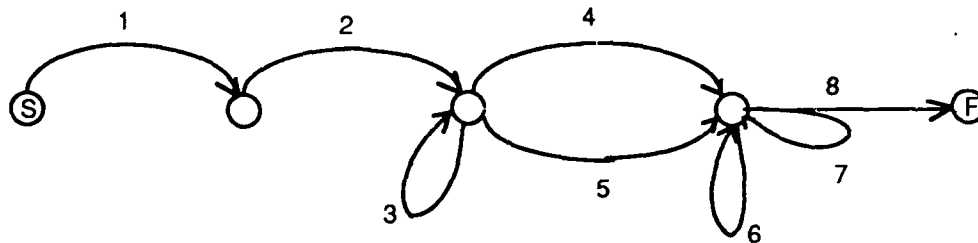
<u>Word</u>	<u>Word Sense</u>	<u>Feature</u>	<u>Feature Value</u>
a	Just-a-Determiner	Part-of-Speech Number	Determiner Singular
bad	Malign	Part-of-Speech	Adjective
been	Def-1	Part-of-Speech Form Type	Verb Past-Participle Be
Ben	Person-Name	Part-of-Speech	Proper-Noun
boats	Group-of-Ships To-Go-Boating	Part-of-Speech Part-of-Speech	Noun Verb
by	Just-a-Preposition	Part-of-Speech	Preposition
Carol	Person-Name	Part-of-Speech	Proper-Noun
caught	To-Have-Captured	Part-of-Speech Form	Verb Past
deadline	Latest-Time	Part-of-Speech	Noun
deer	Friends-of-Rudolph	Part-of-Speech	Noun
entertain	To-Amuse	Part-of-Speech	Verb
firm	Strong	Part-of-Speech	Adjective
given	Def-1	Part-of-Speech Form	Verb Past-Participle
good	Benign	Part-of-Speech	Adjective
green	A-Color	Part-of-Speech	Adjective
have	To-Possess	Part-of-Speech Type	Verb Have
having	Def-1	Part-of-Speech Form	Verb Present-Participle
I	Def-1	Part-of-Speech Number	Pronoun Singular
kumquat	Fruit	Part-of-Speech Number	Noun Singular
kumquats	Fruits	Part-of-Speech Number	Noun Plural
man	Adult-Male To-Staff	Part-of-Speech Part-of-Speech	Noun Verb
old	Ancient Elderly-People	Part-of-Speech Part-of-Speech	Adjective Noun
rabbit	Rodent	Part-of-Speech Number	Noun Singular
rabbits	Rodents	Part-of-Speech Number	Noun Plural
race	Indy-500	Part-of-Speech	Noun

APPENDIX E (Continued)  
A Sample ATN

<u>Word</u>	<u>Word Sense</u>	<u>Feature</u>	<u>Feature Value</u>
run	Operate	Part-of-Speech	Verb
	Move-Quickly	Part-of-Speech	Verb
	A-Race	Part-of-Speech	Noun
the	Just-A-Determiner	Part-of-Speech	Determiner
these	Def-1	Part-of-Speech	Determiner
		Number	Plural
this	Def-1	Part-of-Speech	Determiner
		Number	Singular
to	Just-A-Preposition	Part-of-Speech	Preposition
we Def-1	Part-of-Speech	Pronoun	
		Number	Plural

# APPENDIX E (Continued) A Sample ATN

## b. The S (Sentence) Network:



Local Registers: *Subject, Main-Verb, Auxiliaries, Voice, Direct-Object, and Modifiers.*

Arc	Arc Type	Actions and Conditions	
1	PUSH to NP	Action:	Set Subject to *. (SETR 'SUBJECT *')
2	Verb CAT	Action:	Set Main-Verb to *. (SETR 'MAIN-VERB *')
3	Verb CAT	Condition:	The Type of the Main-Verb is Be or Have. (OR (EQ (GET-F 'TYPE (GET-R 'MAIN-VERB)) 'BE) (EQ (GET-F 'TYPE (GET-R 'MAIN-VERB)) 'HAVE))
		Actions:	Append Main-Verb to Auxiliaries. (ADDR 'AUXILIARIES (GET-R 'MAIN-VERB))
			Set Main-Verb to *. (SETR 'MAIN-VERB *')
4	Verb CAT	Conditions:	The Form of * is Past-Participle. (EQ (GET-F 'FORM) 'PAST-PARTICIPLE)
			The Type of Main-Verb is Be. (EQ (GET-F 'TYPE (GET-R 'MAIN-VERB)) 'BE)
		Actions:	Set Voice to Passive. (SETR 'VOICE 'PASSIVE)
			Append Main-Verb to Auxiliaries. (ADDR 'AUXILIARIES (GET-R 'MAIN-VERB))
			Set Main-Verb to *. (SETR 'MAIN-VERB *')

**APPENDIX E (Continued)**  
**A Sample ATN**

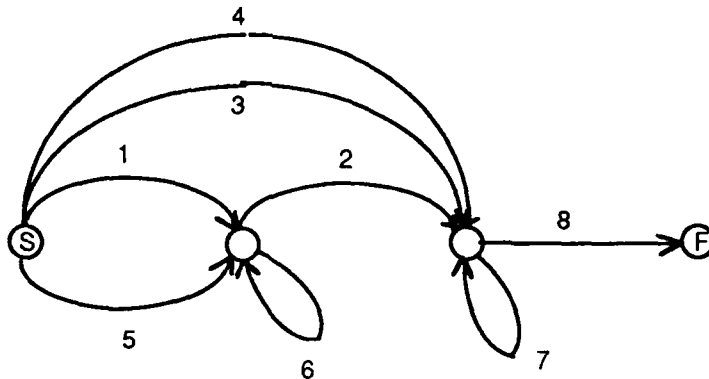
Set Direct-Object to Subject.  
(SETR 'DIRECT-OBJECT (GET-R 'SUBJECT))

Set Subject to a dummy NP.  
(SETR 'SUBJECT' 'DUMMY')

- |    |            |                 |  |
|----|------------|-----------------|--|
| 5. | PUSH to NP | Action:         | Set Direct-Object to *.<br>(SETR 'DIRECT-OBJECT *)   |
| 6. | PUSH to PP | Action:         | Append * to Modifiers.<br>(ADDR 'MODIFIERS *)  |
| 7. | PUSH to PP | Conditions:     | Voice is Passive.<br>(EQ (GET-R 'VOICE) 'PASSIVE))<br><br>Subject is a dummy NP.<br>(EQ (GET-R 'SUBJECT) 'DUMMY)<br><br>The current word of the input is "by".<br>(EQ * 'BY)<br><br>Action: Set Subject to * (the Prep-Object of the PP).<br>(SETR 'SUBJECT *) |
| 8. | POP        | Form to Return: | Representation of the parsed sentence.<br>(BUILDQ '(SENTENCE + (MAIN-VERB +) + + +)<br>'SUBJECT 'MAIN-VERB 'AUXILIARIES<br>'DIRECT-OBJECT 'MODIFIERS)  |

# APPENDIX E (Continued) A Sample ATN

c. The NP (Noun Phrase) Network:



Local Registers: *Number* and *NP*.

<u>Arc</u>	<u>Arc Type</u>	<u>Actions and Conditions</u>
1	Determiner CAT	Action: Set Number to the Number of *. (SETR 'NUMBER (GET-F 'NUMBER))  Add * to NP. (ADDR 'NP *)
2	Noun CAT	Condition: Number is empty or Number is the Number of *. (OR (EQ (GET-R 'NUMBER) NIL) (EQ (GET-R 'NUMBER) (GET-F 'NUMBER)))  Action: Set Number to the Number of *. (SETR 'NUMBER (GET-F 'NUMBER))  Add * to NP. (ADDR 'NP *)
3	Pronoun CAT	Action: Set Number to the Number of *. (SETR 'NUMBER (GET-F 'NUMBER))  Add * to NP. (ADDR 'NP *)
4.	Proper Noun CAT	Action: Set Number to the Number of *. (SETR 'NUMBER (GET-F 'NUMBER))  Add * to NP. (ADDR 'NP *)

APPENDIX E (Continued)  
A Sample ATN

5. JUMP

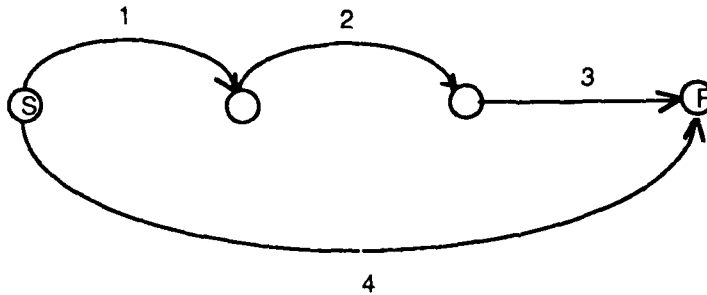
6 Adjective CAT      Action:      Add \* to NP.  
   ADDR 'NP \*')

7 Adjective CAT      Action:      Add \* to NP.  
   (ADDR 'NP \*')

8 POP                      Form to Return: Representation of the parsed Noun Phrase.  
   (BUILDQ '(NP +) 'NP)

# APPENDIX E (Continued) A Sample ATN

## D. The PP (Prepositional Phrase) Network:



Local Registers: *Prepositionr* and *Prep-Object*.

<u>Arc</u>	<u>Arc Type</u>	<u>Actions and Conditions</u>	
1	Preposition CAT	Action:	Set Preposition *. (SETR 'PREPOSITION *')
2	PUSH to NP	Action:	Set Prep-Object to *. (SETR 'PREP-OBJECT *')
3	POP	Form to Return: Representation of the Prepositional Phrase. (BUILDQ '(PP (PREPOSITION +) +) 'PREPOSITION 'PREP-OBJECT)	
4	POP	Condition:	Hold is a PP. (EQ (CAR (GET-R 'HOLD)) 'PP)
		Form to Return: Representation of the Prepositional Phrase. (GET-R 'HOLD)	

## APPENDIX F

### Command Index

Each of the major windows of the ATN Design and Demonstration Tool is made up of two or more functional parts. Below, for each window, the activities of each part are identified and page numbers referencing those activities are given. Also, directions for accessing each major window are provided.

	<u>Pages</u>
<b>Toplevel Window</b>	
To Access From Lisp Listener: <Select>-A	
To Access From Other Tool Windows: Select EXIT	
<b>Command Menu</b>	
DEFINE A NETWORK.....	12
DICTIONARY.....	18-19
READ FROM FILE.....	23, 43 (Appendix E)
WRITE TO FILE.....	22-23
HELP!.....	12
EXIT ATN TOOL.....	
<b>Register Window</b>	
ADD A REGISTER.....	17
Register Name.....	17
VIEW/EDIT.....	17
DELETE.....	17
<b>Network Window</b>	
DELETE TEMPORARIES.....	
Network Name.....	20-21
VIEW/EDIT.....	20-21
DELETE.....	20-21

## APPENDIX F (Continued)

### Command Index

Pages

### Development Windows

To Access From Toplevel Window: Select DEFINE A NETWORK From the Command Menu, or  
Select a Development Network Name from the Network Window.

#### Command Menu

NODES.....	13
Add a Node.....	13
Delete a Node.....	13
Exit NODES.....	13
ARCS.....	13-17
Add an Arc.....	13-14
Reorder Arcs.....	16-17
Remove an Arc.....	14
Exit ARCS.....	13
Defining Arc Features.....	14-16
CHANGE ARC TYPE.....	14
EDIT CONDITIONS.....	14-15
EDIT ACTIONS.....	15
EDIT PRE-ACTIONS.....	16
RUN NETWORK.....	20-22
EXIT.....	12

#### Register Menu

ADD A REGISTER.....	17
Register Name.....	17
VIEW/EDIT.....	17
DELETE.....	17

### Run Windows

To Access From Toplevel Window: Select a Run Network Name from Network Window.

To Access From a Development Window: Select RUN NETWORK From the Command Menu.

#### Command Menu

MODE.....	21
Parse Mode.....	21
Output Mode.....	21-22
Exit.....	21
STEP FORWARD.....	22
CONTINUE.....	22
EXIT.....	20

#### Register Menu

Register Name.....	22
--------------------	----

## APPENDIX F (Continued)

### Command Index

Pages

### Dictionary Window

To Access From Toplevel Window: Select DICTIONARY From Command Menu.

#### Command Menu

CLEAR.....	19
EXIT.....	18

#### Word, Word Sense, and Feature Windows

Adding an Entry.....	18
<i>Word, Word Sense, or Feature Entry</i> .....	18-19
View/Edit an Entry.....	18
Delete an Entry.....	19

### Condition Window

To Access From a Development Window: Select ARCS From the Command Menu, Press Left Mouse-Button Twice While Focused on Arc Identifier Point of an Arc, Select EDIT CONDITIONS From the Menu.

#### Command Menu

ADD A CONDITION.....	15
EXIT.....	15

#### Display Window

<i>Condition Identifier</i> .....	15
VIEW/EDIT.....	15
DELETE.....	15

### Action and Pre-Action Windows

To Access From a Development Window: Select ARCS From the Command Menu, Press Left Mouse-Button Twice While Focused on Arc Identifier Point of an Arc, Select EDIT ACTIONS (or EDIT PRE-ACTIONS) From the Menu.

#### Command Menu

SET A REGISTER.....	16, 41 (Appendix C)
LEFT-ADD TO REGISTER.....	16, 41 (Appendix C)
RIGHT-ADD TO REGISTER.....	16, 41 (Appendix C)
LIFT REGISTER VALUE UP.....	16, 41 (Appendix C)
SEND REGISTER VALUE DOWN.....	16, 41 (Appendix C)
ADD TO HOLD LIST.....	16, 41 (Appendix C)
OTHER ACTION.....	16
EXIT.....	15

#### Display Window

<i>Action or Pre-Action Identifier</i> .....	16
VIEW/EDIT.....	16
DELETE.....	16

## Bibliography

Bates, Madelaine; "The Theory and Practice of Augmented Transition Network Grammars," in Bolc, L. (ed.), Natural Language Communication with Computers, 1978.

Bolc, Leonard (Ed.); The Design of Interpreters, Compilers, and Editors for Augmented Transition Networks, Berlin, Germany: Springer-Verlag, 1983.

Finin, Tim; "A Very Small ATN Interpreter for VanillaLisp," (from the Artificial Intelligence Bulletin Board on the ARPAnet), undated.

Finin, Tim; "The Planes Interpreter and Compiler for Augmented Transition Network Grammars," in Bolc, L. (ed.), Natural Language Communication with Computers, 1978.

Thorne, J., P. Bratley, and H. Dewar; "The Syntactic Analysis of English by Machine," In Michie, D. (ed.), Machine Intelligence, 3, New York: American Elsevier, 1968.

Winograd, Terry; Language as a Cognitive Process, Reading, MA: Addison-Wesley, 1983.

Woods, William A.; "Transition Network Grammars for Natural Language Analysis," Communications of the ACM, Vol. 13, Number 10, October 1970.